

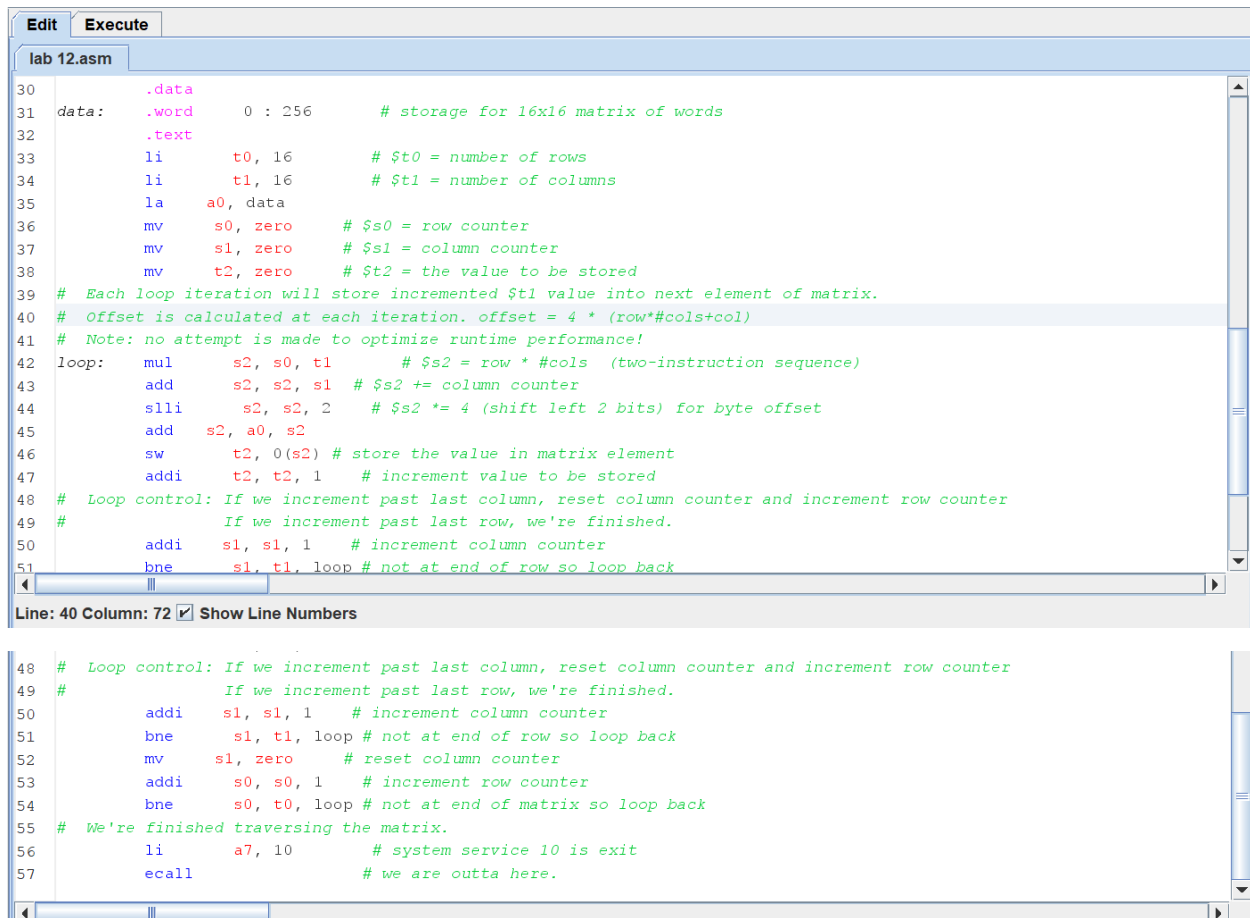
Student name: Lê Ngọc Anh Vũ

Student ID: 20236014

## Lab 12

### Assignment 1 - Running the Data Cache Simulator tool

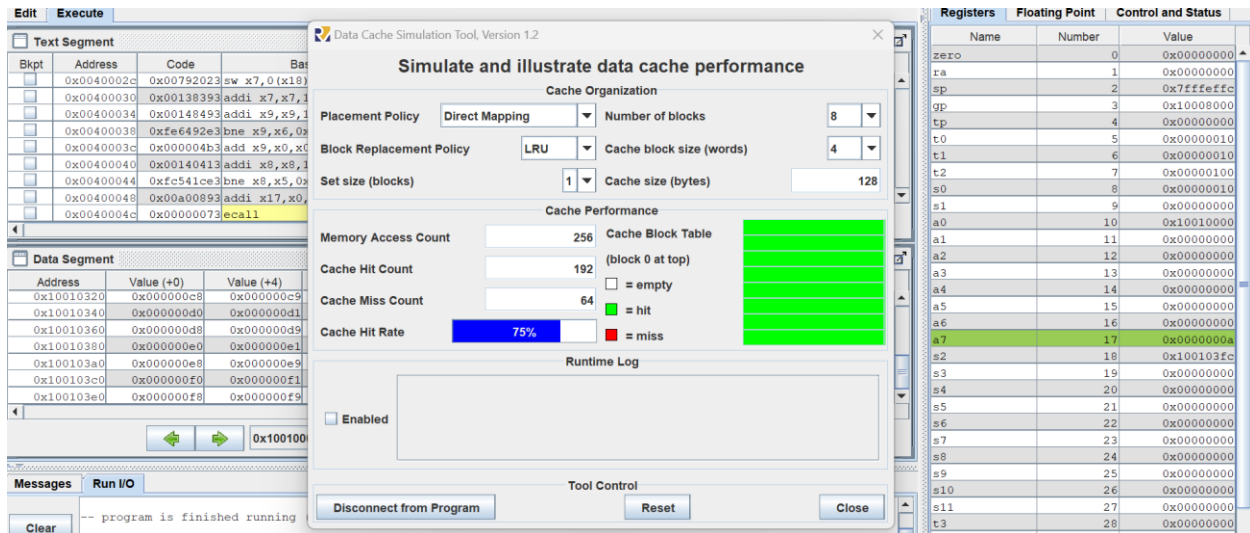
Open the program row-major.asm:



```
30      .data
31 data: .word    0 : 256      # storage for 16x16 matrix of words
32      .text
33      li        t0, 16      # $t0 = number of rows
34      li        t1, 16      # $t1 = number of columns
35      la        a0, data
36      mv        s0, zero     # $s0 = row counter
37      mv        s1, zero     # $s1 = column counter
38      mv        t2, zero     # $t2 = the value to be stored
39 # Each loop iteration will store incremented $t1 value into next element of matrix.
40 # Offset is calculated at each iteration. offset = 4 * (row*#cols+col)
41 # Note: no attempt is made to optimize runtime performance!
42 loop: mul       s2, s0, t1   # $s2 = row * #cols (two-instruction sequence)
43      add       s2, s2, s1    # $s2 += column counter
44      slli      s2, s2, 2     # $s2 *= 4 (shift left 2 bits) for byte offset
45      add       s2, a0, s2
46      sw        t2, 0(s2)    # store the value in matrix element
47      addi      t2, t2, 1     # increment value to be stored
48 # Loop control: If we increment past last column, reset column counter and increment row counter
49 # If we increment past last row, we're finished.
50      addi      s1, s1, 1     # increment column counter
51      bne       s1, t1, loop # not at end of row so loop back
52
53      addi      s1, s1, 1     # increment column counter
54      bne       s1, t1, loop # not at end of row so loop back
55      mv        s1, zero     # reset column counter
56      addi      s0, s0, 1     # increment row counter
57      bne       s0, t0, loop # not at end of matrix so loop back
58
59      li        a7, 10       # system service 10 is exit
60      ecall
61
62 # We're finished traversing the matrix.
63 # we are outta here.
```

Run the program with run speed slider is 30 instructions per second.

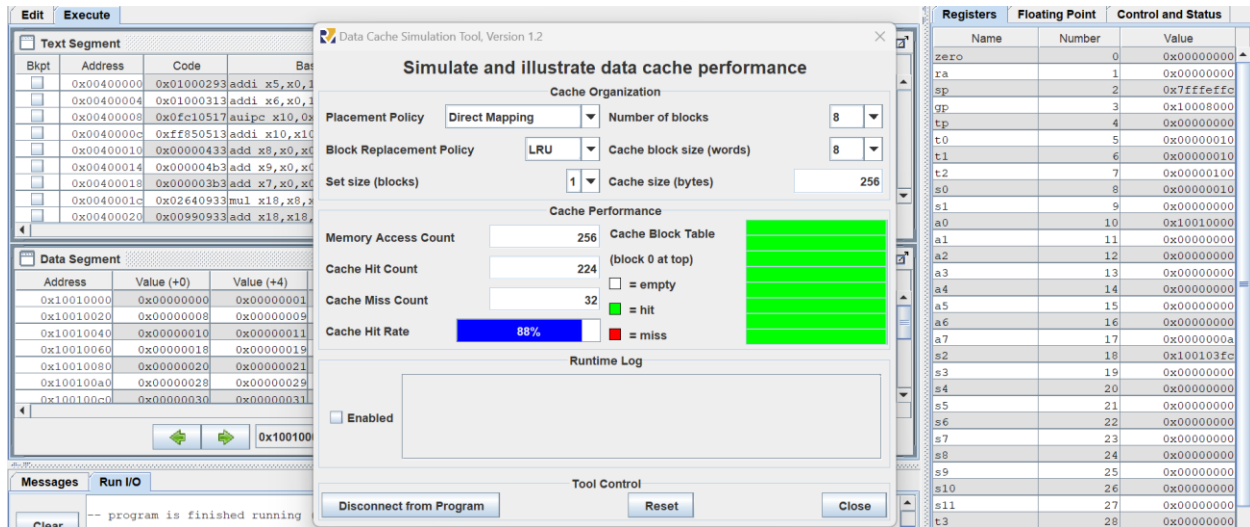
We got the final cache hit rate which was 75% as the picture below . With each miss, a block of 4 words are written into the cache. In a row-major traversal, matrix elements are accessed in the same order they are stored in memory. Thus, each cache miss is followed by 3 hits as the next 3 elements are found in the same cache block. This is followed by another miss when Direct Mapping maps to the next cache block, and the patterns repeats itself. So, 3 of every 4 memory accesses will be resolved in cache.



- For the block size is increased from 4 words to 8 words:

The final cache hit rate will be: 87,5% or rounding to 88%

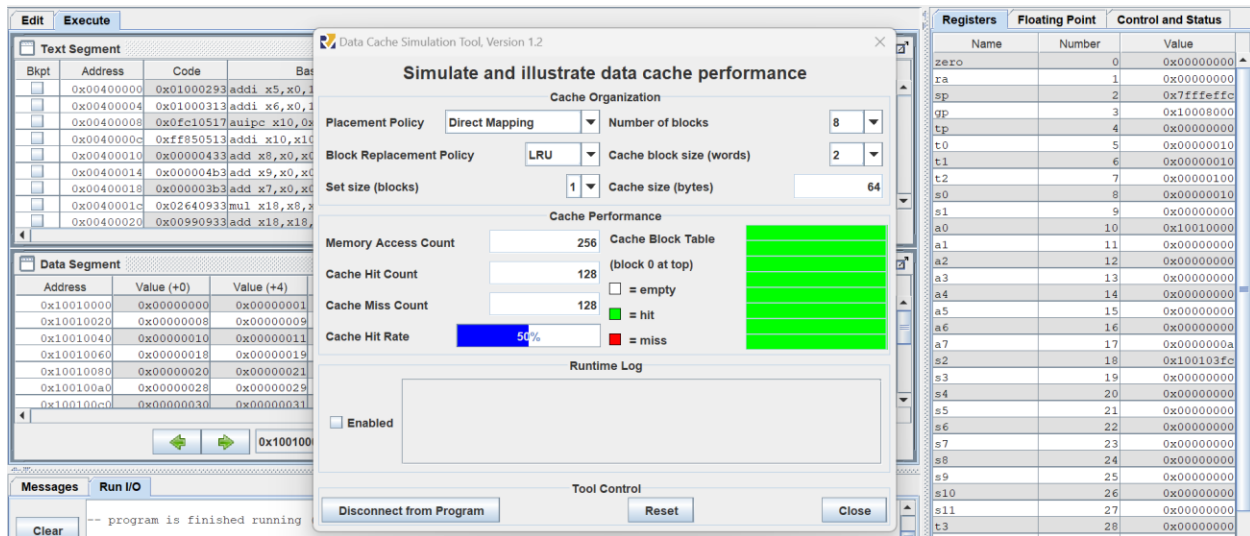
Check for the prediction above:



- For the block size is decreased from 4 words to 2 words:

The final cache hit rate will be: 50%

Check for the prediction above:



Open the program column-major.asm:

```

29 # 3. The "li" to initialize $t1 needs to be changed to the new #columns.
30 #
31 .data
32 data: .word 0 : 256 # 16x16 matrix of words
33 .text
34 li t0, 16 # $t0 = number of rows
35 li t1, 16 # $t1 = number of columns
36 la a0, data
37 mv s0, zero # $s0 = row counter
38 mv s1, zero # $s1 = column counter
39 mv t2, zero # $t2 = the value to be stored
40 # Each loop iteration will store incremented $t1 value into next element of matrix.
41 # Offset is calculated at each iteration. offset = 4 * (row*cols+col)
42 # Note: no attempt is made to optimize runtime performance!
43 loop: mul s2, s0, t1 # $s2 = row * #cols (two-instruction sequence)
44 add s2, s2, s1 # $s2 += col counter
45 slli s2, s2, 2 # $s2 *= 4 (shift left 2 bits) for byte offset
46 add s2, a0, s2
47 sw t2, 0(s2) # store the value in matrix element
48 addi t2, t2, 1 # increment value to be stored
49 # Loop control: If we increment past bottom of column, reset row and increment column
50 # If we increment past the last column, we're finished.
51 #
52 # Loop control: If we increment past bottom of column, reset row and increment column
53 # If we increment past the last column, we're finished.
54 addi s0, s0, 1 # increment row counter
55 bne s0, t0, loop # not at bottom of column so loop back
56 mv s0, zero # reset row counter
57 addi s1, s1, 1 # increment column counter
58 bne s1, t1, loop # loop back if not at end of matrix (past the last column)
59 # We're finished traversing the matrix.
60 li a7, 10 # system service 10 is exit
61 ecall # we are outta here.

```

Run the program with run speed slider is 30 instructions per second.

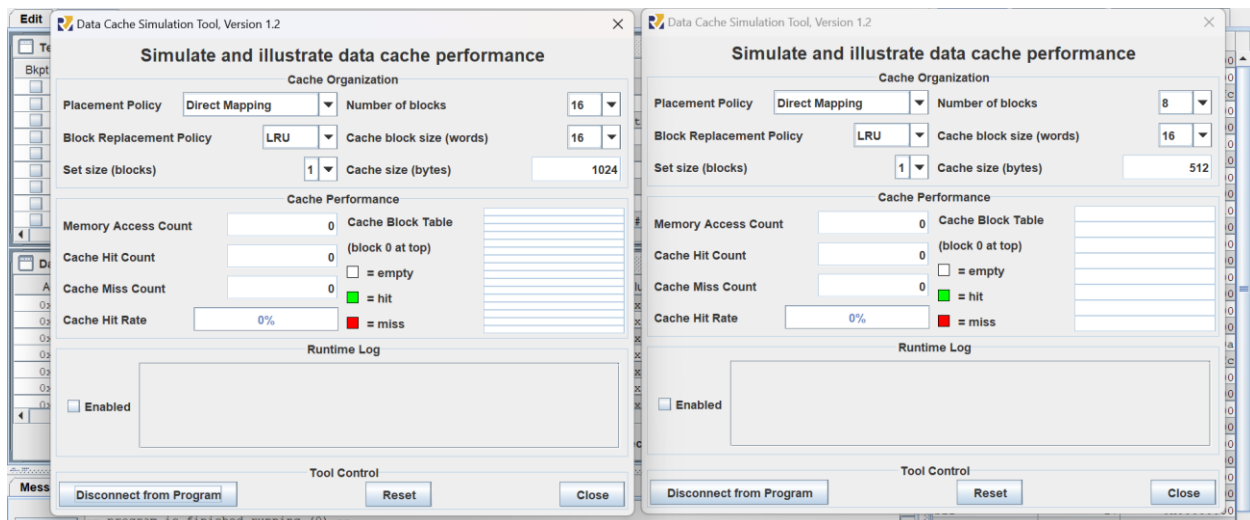
We got the cache performance for this program which was 0% as the picture below. The problem is the memory locations are now accessed not sequentially as before, but each access is 16 words beyond the previous one (circularly). With the settings we've used, no two consecutive memory accesses occur in the same block, so every access is a miss.

The screenshot shows the 'Data Cache Simulation Tool, Version 1.2' window. The 'Cache Organization' section has 'Placement Policy' set to 'Direct Mapping', 'Number of blocks' set to 8, 'Block Replacement Policy' set to 'LRU', 'Cache block size (words)' set to 4, and 'Cache size (bytes)' set to 128. The 'Cache Performance' section shows 'Memory Access Count' as 256, 'Cache Hit Count' as 0, 'Cache Miss Count' as 256, and 'Cache Hit Rate' as 0%. The 'Cache Block Table' shows 8 blocks, all of which are red, indicating they are empty. The 'Runtime Log' is empty. The 'Tool Control' section has 'Disconnect from Program', 'Reset', and 'Close' buttons. The background shows a code editor with assembly code and a register window on the right.

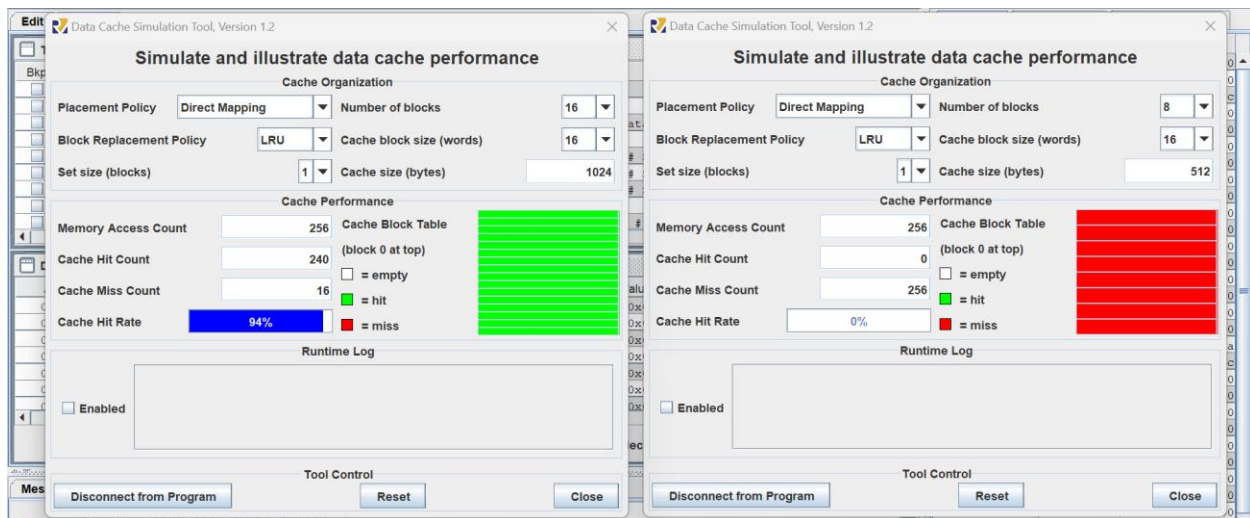
Change the block size to 16:

The screenshot shows the 'Data Cache Simulation Tool, Version 1.2' window with the 'Cache block size (words)' set to 16. The 'Cache size (bytes)' is now 512. The 'Cache Performance' section shows 'Memory Access Count' as 0, 'Cache Hit Count' as 0, 'Cache Miss Count' as 0, and 'Cache Hit Rate' as 0%. The 'Cache Block Table' shows 8 blocks, all of which are white, indicating they are empty. The 'Runtime Log' is empty. The 'Tool Control' section has 'Disconnect from Program', 'Reset', and 'Close' buttons. The background shows the same code editor and register window as the previous screenshot.

Create a second instance of the Cache Simulator. Connect the new tool instance to RISC-V, change its block size to 16 and change its number of blocks to 16.



- The cache performance of the original tool instance is still 0%. Block size 16 didn't help because there was still only one access to each block, the initial miss, before that block was replaced with a new one.
- The cache performance of the second tool instance is 94%. At this point, the entire matrix will fit into cache and so once a block is read in it is never replaced. Only the first access to a block results in a miss.



## Assignment 2 – Running the Memory Reference Visualization tool

Open the program row-major.asm:

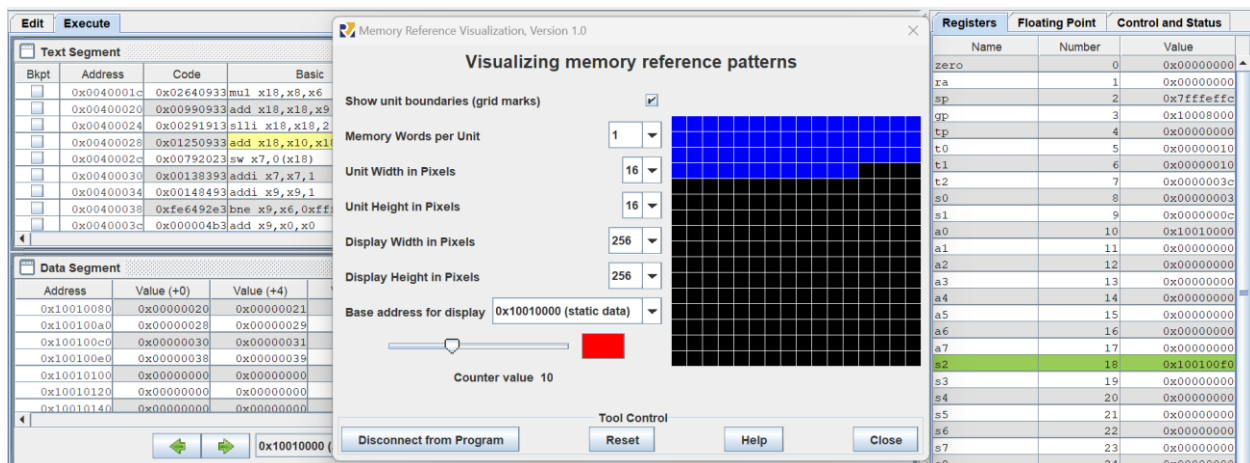
```

Edit Execute
lab 12.asm
30      .data
31 data: .word      0 : 256      # storage for 16x16 matrix of words
32      .text
33      li      t0, 16          # $t0 = number of rows
34      li      t1, 16          # $t1 = number of columns
35      la      a0, data
36      mv      s0, zero        # $s0 = row counter
37      mv      s1, zero        # $s1 = column counter
38      mv      t2, zero        # $t2 = the value to be stored
39 # Each loop iteration will store incremented $t1 value into next element of matrix.
40 # Offset is calculated at each iteration. offset = 4 * (row*cols+col)
41 # Note: no attempt is made to optimize runtime performance!
42 loop: mul      s2, s0, t1      # $s2 = row * #cols (two-instruction sequence)
43      add      s2, s2, s1      # $s2 += column counter
44      slli     s2, s2, 2       # $s2 *= 4 (shift left 2 bits) for byte offset
45      add      s2, a0, s2
46      sw      t2, 0(s2)      # store the value in matrix element
47      addi     t2, t2, 1      # increment value to be stored
48 # Loop control: If we increment past last column, reset column counter and increment row counter
49 # If we increment past last row, we're finished.
50      addi     s1, s1, 1      # increment column counter
51      bne      s1, t1, loop   # not at end of row so loop back
52
53      addi     s0, s0, 1      # increment row counter
54      bne      s0, t0, loop   # not at end of matrix so loop back
55 # We're finished traversing the matrix.
56      li      a7, 10          # system service 10 is exit
57      ecall

```

Line: 40 Column: 72 ☒ Show Line Numbers

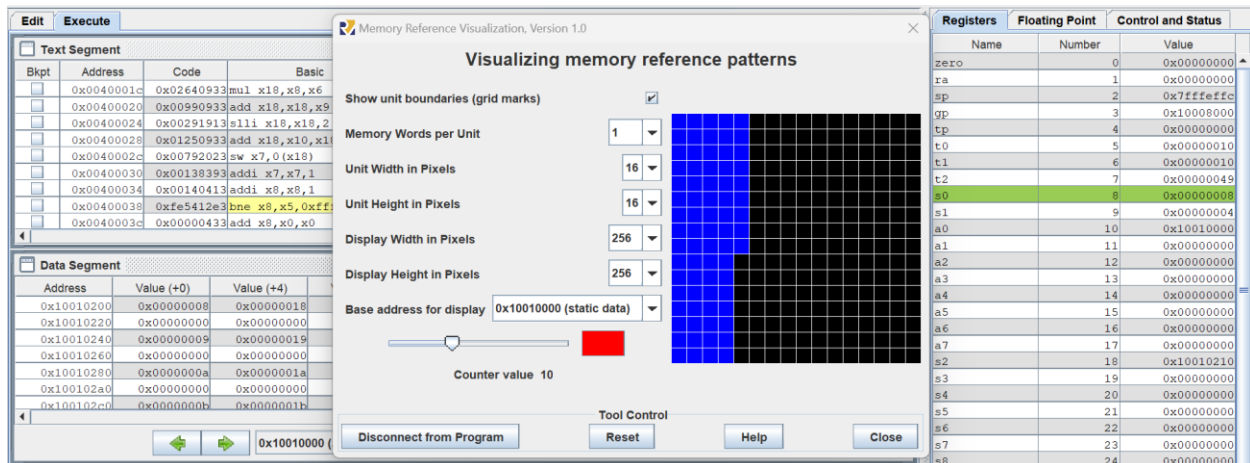
Run the program with run speed slider which is 30 instructions per second. The grid unit is colored sequentially in row, from left to right, then next row down.



For the program column-major.asm:

The grid unit is colored respectively in column, from up to down, then next right column





For the program Fibonacci.asm:

There are a lot of memory words which are referenced many times because the characteristic of Fibonacci:  $\text{fib}[n] = \text{fib}[n-1] + \text{fib}[n-2]$ . It means that, in order to compute  $n$ -th Fibonacci number, we have to access to before memory words.

